Refine Search

Search Results -

Terms	Documents
L2 same (microprocessor or processor)	5

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L3

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Monday, June 21, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,USOC; PLUR=YES; OP=OR		
L3	L2 same (microprocessor or processor)	5	L3
L2	L1 same (slave or (I adj1 O) or (input adj1 output))	23	L2
L1	(transaction or task or job) near3 reorder$3	189	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

Refine Search

Search Results -

Terms	Documents
L3	0

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L4

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Monday, June 21, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L4	L3	0	L4
	DB=USPT,USOC; PLUR=YES; OP=OR		
L3	L2 same (microprocessor or processor)	5	L3
L2	L1 same (slave or (I adj1 O) or (input adj1 output))	23	L2
L1	(transaction or task or job) near3 reorder$3	189	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

Refine Search

Search Results -

Terms	Documents
(709/100 |709/208 |710/110 |710/107 |710/263 |710/41 |710/52 |710/311 |711/151 |714/47).ccls.	4004

Database: US Pre-Grant Publication Full-Text Database; US Patents Full-Text Database; US OCR Full-Text Database; EPO Abstracts Database; JPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Search: L5

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Monday, June 21, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,USOC; PLUR=YES; OP=OR		
L5	710/110,107,263,41,52,311;709/100,208;714/47;711/151.ccls.	4004	L5
	DB=EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L4	L3	0	L4
	DB=USPT,USOC; PLUR=YES; OP=OR		
L3	L2 same (microprocessor or processor)	5	L3
L2	L1 same (slave or (I adj1 O) or (input adj1 output))	23	L2
L1	(transaction or task or job) near3 reorder$3	189	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

Refine Search

Search Results -

Terms	Documents
L3 or L6	11

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L7

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Monday, June 21, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT,USOC; PLUR=YES; OP=OR			
L7	l3 or L6	11	L7
L6	l2 and L5	9	L6
L5	710/110,107,263,41,52,311;709/100,208;714/47;711/151.ccls.	4004	L5
DB=EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L4	L3	0	L4
DB=USPT,USOC; PLUR=YES; OP=OR			
L3	L2 same (microprocessor or processor)	5	L3
L2	L1 same (slave or (I adj1 O) or (input adj1 output))	23	L2
L1	(transaction or task or job) near3 reorder$3	189	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

EAST - [Untitled1:1]

File View Edit Tools Window Help

- Drafts
- Pending
- Active
  - L1: (254) (transaction or
  - L2: (44) l1 same (microproc
  - L3: (5) l2 same queu$3
- Failed
- Saved
- Favorites
- Tagged (0)
- UDC
- Queue
- Trash

Search | List | Browse | Queue | Clear

DBs | USPAT | Plurals

Default operator: OR | Highlight all hit terms initially

BRS I... | IS&R ... | Image | Text | HTML

	Type	L #	Hits	Search Text	DBs	Time Stamp	Comments	Error Definition	Err
1	BRS	L1	254	(transaction or task or job) near5 reorder$3	USPAT	2004/06/21 15:32			0
2	BRS	L2	44	l1 same (microprocessor or processor)	USPAT	2004/06/21 15:33			0
3	BRS	L3	5	l2 same queu$3	USPAT	2004/06/21 15:33			0

Start | EAST - [Untitled1...

EAST - [Untitled1:1]

File View Edit Tools Window Help

- Drafts
- Pending
- Active
  - L1: (254) (transaction or
  - L2: (44) l1 same (micropro
  - L3: (5) 12 same queu$3
- Failed
- Saved
- Favorites
- Tagged (0)
- UDC
- Queue
- Trash

Search | List | Browse | Queue | Clear

DBs | USPAT | Plurals

Default operator: OR | Highlight all hit terms initially

12 same queu$3

BRS f... | S&R... | Image | Text | HTML

	U	1	Document ID	Issue Date	Pages	Title	Current OR	Current XRef	R
1	☐	☐	US RE38428 E	20040210	37	Bus transaction reordering in a computer system having	710/110	370/402; 709/208;	
2	☐	☐	US 6505259 B1	20030107	7	Reordering of burst data transfers across a host	710/35	710/20; 710/52;	
3	☐	☐	US 6246993 B1	20010612	75	Reorder system for use with an electronic printing press	705/9	700/100; 700/110;	
4	☐	☐	US 5617556 A	19970401	10	System and method to prevent the occurrence of a snoop	711/141	711/118	
5	☐	☐	US 5293620 A	19940308	16	Method apparatus for scheduling tasks in repeated	718/102		

Start | EAST - [Untitled1:...

Your search matched **2** of **1046194** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance**
Descending order.

Refine This Search:
You may refine your search by editing the current search expression or enter
new one in the text box.

(transaction or task or job)<and>reorder* and (proce Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

---

1 **Efficient job scheduling in a mesh multicomputer without discrimin against large jobs**
Dugki Min; Mutka, M.W.;
Parallel and Distributed Processing, 1995. Proceedings. Seventh IEEE Sympos on , 25-28 Oct. 1995
Pages:52 - 59

[Abstract] [PDF Full-Text (592 KB)] **IEEE CNF**

---

2 **Designing a modern memory hierarchy with hardware prefetching**
Wei-Fen Lin; Reinhardt, S.K.; Burger, D.;
Computers, IEEE Transactions on , Volume: 50 , Issue: 11 , Nov. 2001
Pages:1202 - 1218

[Abstract] [PDF Full-Text (1732 KB)] **IEEE JNL**

---

h eee e eee g e ch e ch e e c e e e c e

Efficient job scheduling in a mesh multicomputer w discrimination against large jobs

Dugki Min Mutka, M.W.
Dept. of Comput. Sci., Kon-Kuk Univ., Seoul, South Korea;

This paper appears in: **Parallel and Distributed Processing, 1995. Proce Seventh IEEE Symposium on**

Meeting Date: 10/25/1995 - 10/28/1995
Publication Date: 25-28 Oct. 1995
Location: San Antonio, TX USA
On page(s): 52 - 59
Reference Cited: 11
Inspec Accession Number: 5119665

Abstract:
Many innovative schemes for allocating jobs to parallel computing systems ha proposed in order to achieve highly utilized parallel computing systems. The s have tried to achieve good **job** response times with little system fragmentatic processing resources. Since most schemes have concentrated on approaches **processor** allocation, the schemes have used First-Come-First-Serve (FCFS) scheduling discipline. However, it has been previously established that **job** sc algorithms for parallel computing systems can have a large impact on the sys utilization and **job** response time. Schemes that use multiple **queues**, which sequence of jobs allocated to the parallel system, can be very effective in imp system performance. However, such non-FCFS schemes have been criticized they provide improved average performance by favoring small jobs at the exp large jobs. In order to achieve improved performance by means of multiple **q** scheduling schemes without sacrificing the fairness of FCFS, we propose a ne scheduling discipline that behaves in a FCFS manner under low loaded conditi exploits performance enhancing features of multiple **queue** schemes under hi conditions. In addition, the scheme does not inappropriately discriminate agai jobs

Index Terms:
First-Come-First-Serve discrimination **job** allocation **job** response time **job** respons scheduling **job** scheduling algorithms large jobs mesh multicomputer multiple **queu** scheduling schemes multiple **queues** multiprocessing systems parallel computing sy

h eee e eee g e ch ch b c be be

performance evaluation **processor** allocation **processor** scheduling resource alloca scheduling small jobs system fragmentation system performance system utilization First-Serve discrimination **job** allocation **job** response time **job** response times **jo** **job** scheduling algorithms large jobs mesh multicomputer multiple **queue job** schedu schemes multiple **queues** multiprocessing systems parallel computing systems per evaluation **processor** allocation **processor** scheduling resource allocation schedul jobs system fragmentation system performance system utilization

Documents that cite this document
There are no citing documents available in IEEE Xplore at this time.

Search Results [PDF FULL-TEXT 592 KB] NEXT DOWNLOAD CITATION

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Search | Join IEEE | Web Account | New this week | OPAC Linking Information | Your Feedback | Technical Support | Email Alerting | No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ| Terms | Back to Top

h eee e eee g e ch ch b c be be

US-PAT-NO: 5293620

DOCUMENT-IDENTIFIER: US 5293620 A

TITLE: Method apparatus for scheduling tasks in repeated iterations in a digital data processing system having multiple processors

---------- KWIC ---------

Detailed Description Text - DETX (20):
As noted above, in connection with step 116 (FIG. 2B), the task dispatcher 12 may, after (1) receiving the notification from a processor 10 that it has finished processing its assigned task and (2) determining that processing of tasks pointed to by all of the entries 15 has been completed, reorder the entries 15 in the task identification queue 13 so that one or more of the entries 15 relating to the last completed tasks are moved to the beginning of the queue 13, so that they may be used to dispatch tasks at the beginning of the next iteration. In the following, it will be assumed that only one entry 15, which points to the last-completed task, is moved to the beginning of the task identification queue 13. In this operation, the task dispatcher first retrieves the contents of the entry in the task assignment list 16 associated with the processor 10 from which it received the notification (step 120). The contents retrieved from the task assignment list 16 points to the entry 15(i) in the task identification queue 13 that, in turn, identified the last finished task in the task store 11. The task dispatcher 12 then dequeues the entry 15(i) in the task identification queue 13 identified by the contents retrieved in step 120 (step 121) and enqueues it at the head of the task identification queue 13 (step 122).

Claims Text - CLTX (4):
C. a task dispatcher comprising means for sequentially dispatching the plurality of tasks to said plurality of digital data processors by sequentially referring to the plurality of task identification entries in the order that the plurality of task identification entries are arranged in said task identification queue to retrieve the plurality of tasks, and means for reordering the plurality of task identification entries in said task identification queue after all of the tasks represented by the plurality of

1 of 1 DOCUMENT

UNITED STATES PATENT AND TRADEMARK OFFICE GRANTED PATENT

5996036

Link to Claims Section

November 30, 1999

Bus transaction reordering in a computer system having unordered slaves

REISSUE: November 30, 2001 - Reissue Application filed Ex. Gp.: 2181; Re. S.N. 10/006,939 (O.G. June 18, 2002)
September 22, 2003 - Reissue Application filed Ex. Gp.: 2728; Re. S.N. 10/669,119 (O.G. December 2, 2003)
February 10, 2004 - This Patent was reissued as Reissue Patent RE 38,428 (O.G. February 10, 2004)

APPL-NO: 779632 (08)

FILED-DATE: January 7, 1997

GRANTED-DATE: November 30, 1999

ASSIGNEE-AT-ISSUE: Apple Computers, Inc., Cupertino, California, United States (US), 02

ASSIGNEE-AFTER-ISSUE: April 2, 1999 - ASSIGNMENT OF ASSIGNORS INTEREST (SEE DOCUMENT FOR DETAILS)., APPLE COMPUTER, INC. ONE INFINITE LOOP CUPERTINO CALIFORNIA 95014, Reel and Frame Number: 009859/0365

ENGLISH-ABST:

A mechanism is provided for reordering bus transactions to increase bus utilization in a computer system in which a split- transaction bus is bridged to a single-envelope bus. In one embodiment, both masters and slaves are ordered, simplifying implementation. In another embodiment, the system is more loosely coupled with only masters being ordered. Greater bus utilization is thereby achieved. To avoid deadlock, transactions begun on the split-transaction bus are monitored. When a combination of transactions would, if a predetermined further transaction were to begin, result in deadlock, this condition is detected. In the more tightly coupled system, the predetermined further transaction, if it is requested, is refused, thereby avoiding deadlock. In the more loosely- coupled system, the flexibility afforded by unordered slaves is taken advantage of to, in the typical case, reorder the transactions and avoid deadlock without killing any transaction. Where a data dependency exists that would prevent such reordering, the further transactions is killed as in the more tightly-coupled embodiment. Data dependencies are detected in accordance with address-coincidence signals generated by slave devices on a cache-line basis. In accordance with a further optimization, at least one slave device (e.g., DRAM) generates page-coincidence bits. When two transactions to the slave device are to the same address page, the transactions are reordered if necessary to ensure that they are executed one after another without any intervening transaction. Latency of the slave is thereby reduced.

No Documents Found!

No documents were found for your search (**5996036 or 5,996,036**). Please edit your search and try again. You may want to try one or more of the following:

- Check for spelling errors.
- Remove some search terms.
- Use more common search terms.
- If applicable, look for all dates.

Edit Search

LEXIS-NEXIS
Library: PATENTS
File: CASES

No Documents Found!

No documents were found for your search (**5996036 or 5,996,036**). Please edit your search and try again. You may want to try one or more of the following:

- Check for spelling errors.
- Remove some search terms.
- Use more common search terms.
- If applicable, look for all dates.

Edit Search

No Documents Found!

No documents were found for your search (**5996036 or 5,996,036**). Please edit your search and try again. You may want to try one or more of the following:

- Check for spelling errors.
- Remove some search terms.
- Use more common search terms.
- If applicable, look for all dates.

Edit Search

?us5996036/pn

** SS 1: Results 1

Search statement 2

?prt full nonstop legalall

1/1 PLUSPAT - (C) QUESTEL-ORBIT- image
PN - US5996036 A 19991130 [US5996036]
TI - (A) Bus transaction reordering in a computer system having unordered slaves
PA - (A) APPLE COMPUTERS INC (US)
PA0 - Apple Computers, Inc., Cupertino CA [US]
IN - (A) KELLY JAMES D (US)
AP - US77963297 19970107 [1997US-0779632]
PR - US77963297 19970107 [1997US-0779632]
IC - (A) G06F-009/46 G06F-013/36 G11C-007/00
EC - G06F-013/36L
- G06F-013/40D1R
PCL - ORIGINAL (O) : 710110000; CROSS-REFERENCE (X) : 709208000 710107000
DT - Basic
CT - US4181974; US4473880; US4965716; US5006982; US5191649; US5257356; US5287477; US5327538; US5345562; US5375215; US5473762; US5592631; US5682512; US5822772
STG - (A) United States patent
AB - A mechanism is provided for reordering bus transactions to increase bus utilization in a computer system in which a split-transaction bus is bridged to a single-envelope bus. In one embodiment, both masters and slaves are ordered, simplifying implementation. In another embodiment, the system is more loosely coupled with only masters being ordered. Greater bus utilization is thereby achieved. To avoid deadlock, transactions begun on the split-transaction bus are monitored. When a combination of transactions would, if a predetermined further transaction were to begin, result in deadlock, this condition is detected. In the more tightly coupled system, the predetermined further transaction, if it is requested, is refused, thereby avoiding deadlock. In the more loosely-coupled system, the flexibility afforded by unordered slaves is taken advantage of to, in the typical case, reorder the transactions and avoid deadlock without killing any transaction. Where a data dependency exists that would prevent such reordering, the further transactions is killed as in the more tightly-coupled embodiment. Data dependencies are detected in accordance with address-coincidence signals generated by slave devices on a cache-line basis. In accordance with a further optimization, at least one slave device (e.g., DRAM) generates page-coincidence bits. When two transactions to the slave device are to the same address page, the transactions are reordered if necessary to ensure that they are executed one after another without any intervening transaction. Latency of the slave is thereby reduced.

1/1 LGST - (C) EPO
PN - US5996036 A 19991130 [US5996036]
AP - US77963297 19970107 [1997US-0779632]
ACT - 20020618 US/RF-A
REISSUE APPLICATION FILED
EFFECTIVE DATE: 20011130
- 20031202 US/RF-A
REISSUE APPLICATION FILED
EFFECTIVE DATE: 20030922
UP - 2003-51

1/1 CRXX - (C) CLAIMS/RRX
PN - 5,996,036 A 19991130 [US5996036]
PA - Apple Computer Inc
ACT - 20011130 REISSUE REQUESTED
ISSUE DATE OF O.G.: 20020618
REISSUE REQUEST NUMBER: 10/006939
EXAMINATION GROUP RESPONSIBLE FOR REISSUEPROCESS: 2181
Reissue Patent Number: USRE38428

- 20030922 REISSUE REQUESTED
ISSUE DATE OF O.G.: 20031202
REISSUE REQUEST NUMBER: 10/669119
EXAMINATION GROUP RESPONSIBLE FOR REISSUEPROCESS: 2728

Reissue Patent Number:

First Hit Fwd Refs

Generate Collection Print

L7: Entry 1 of 11 File: USPT Apr 20, 2004

US-PAT-NO: 6725297
DOCUMENT-IDENTIFIER: US 6725297 B1

TITLE: Peripheral interface circuit for an I/O node of a computer system

DATE-ISSUED: April 20, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Askar; Tahsin	Round Rock	TX		
Hewitt; Larry D.	Austin	TX		
Chambers; Eric G.	Austin	TX		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 10/ 093146 [PALM]
DATE FILED: March 7, 2002

PARENT-CASE:
This is a continuation-in-part of application Ser. No. 09/978,534 filed on Oct. 15, 2001.

INT-CL: [07] G06 F 13/00

US-CL-ISSUED: 710/52; 710/5, 710/8, 710/11, 710/12, 710/20, 710/21, 710/65
US-CL-CURRENT: 710/52; 710/11, 710/12, 710/20, 710/21, 710/5, 710/65, 710/8

FIELD-OF-SEARCH: 710/5, 710/8, 710/11, 710/12, 710/20, 710/21, 710/52, 710/65

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5187780	February 1993	Clark et al.	
☐	5717878	February 1998	Sannino	725/117
☐	6278532	August 2001	Heimendinger et al.	
☐	6414525	July 2002	Urakawa	

h e b b g e e e f c e c ghe e ge

☐	6414961	July 2002	Katayanagi	
☐	6487628	November 2002	Duong et al.	710/313

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
98/14015	April 1998	WO	

OTHER PUBLICATIONS

International Search Report, Application No. PCT/US 02/26884, mailed Dec. 20, 2002.
U.S. patent application Ser. No. 09/399,281, filed Sep. 17, 1999.

ART-UNIT: 2182

PRIMARY-EXAMINER: Gaffin; Jeffrey

ASSISTANT-EXAMINER: Farooq; Mohammad O.

ATTY-AGENT-FIRM: Meyertons Hood Kivlin Kowert & Goetzel, P.C. Kivlin; B. Noel

ABSTRACT:

A peripheral interface circuit for an I/O node of a computer system. A peripheral interface circuit for an input/output node of a computer system includes a first buffer circuit, a second buffer circuit and a bus interface circuit. The first buffer circuit receives packet commands and may include a first plurality of buffers each corresponding to a respective virtual channel of a plurality of virtual channels. The second buffer circuit is coupled to receive packet commands from the bus interface circuit and may include a second plurality of buffers each corresponding to a respective virtual channel of the plurality of virtual channels. The bus interface circuit may be configured to translate selected packet commands stored in the first buffer circuit into commands suitable for transmission on a peripheral bus.

27 Claims, 13 Drawing figures

h e b b g e e e f c e c ghe e ge

First Hit Fwd Refs

Generate Collection Print

L7: Entry 4 of 11 File: USPT Feb 10, 2004

US-PAT-NO: RE38428
DOCUMENT-IDENTIFIER: US RE38428 E

TITLE: Bus transaction reordering in a computer system having unordered slaves

DATE-ISSUED: February 10, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Kelly; James D.	Scotts Valley	CA		
Regal; Michael L.	Pleasanton	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Apple Computer, Inc.	Cupertino	CA			02

APPL-NO: 10/ 006939 [PALM]
DATE FILED: November 30, 2001

REISSUE-DATA:

US-PAT-NO	DATE-ISSUED	APPL-NO	DATE-FILED
05996036	November 30, 1999	779632	January 7, 1997

PARENT-CASE:
.Iadd.This application is a continuation-in-part of U.S. patent application Ser. No. 08/432,622, filed May 2, 1995, now abandoned..Iaddend.

INT-CL: [07] G06 F 9/46, G06 F 13/36, G11 C 7/00

US-CL-ISSUED: 710/110; 710/107, 709/208, 370/402
US-CL-CURRENT: 710/110; 370/402, 709/208, 710/107

FIELD-OF-SEARCH: 710/110, 710/107, 710/263, 710/41, 710/52, 710/311, 709/100, 709/208, 714/47, 711/151, 370/402

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4181974	January 1980	Lemay et al.	

h e b b g e e e f c e c ghe e ge

☐	4473880	September 1984	Budde et al.
☐	4494193	January 1985	Brahm et al.
☐	4965716	October 1990	Sweeney
☐	5006982	April 1991	Ebersole et al.
☐	5191649	March 1993	Cadambi et al.
☐	5257356	October 1993	Brockmann et al.
☐	5287477	February 1994	Johnson et al.
☐	5305442	April 1994	Pedersen et al.
☐	5307505	April 1994	Houlberg et al.
☐	5327538	July 1994	Hamaguchi et al.
☐	5327570	July 1994	Foster et al.
☐	5333276	July 1994	Solari
☐	5345562	September 1994	Chen
☐	5355455	October 1994	Hilgendorf et al.
☐	5363485	November 1994	Nguyen et al.
☐	5369748	November 1994	McFarland et al.
☐	5375215	December 1994	Hanawa et al.
☐	5418914	May 1995	Heil et al.
☐	5442763	August 1995	Bartfai et al.
☐	5469435	November 1995	Krein et al.
☐	5473762	December 1995	Krein et al.
☐	5542056	July 1996	Jaffa et al.
☐	5544332	August 1996	Chen
☐	5546546	August 1996	Bell et al.
☐	5592631	January 1997	Kelly et al.
☐	5592670	January 1997	Pletcher
☐	5615343	March 1997	Sarangdhar et al.
☐	5680402	October 1997	Olnowich et al.
☐	5682512	October 1997	Tetrick
☐	5708794	January 1998	Parks et al.
☐	5822772	October 1998	Chan et al.
☐	5930485	July 1999	Kelly
☐	5933612	August 1999	Kelly et al.

ART-UNIT: 2181

PRIMARY-EXAMINER: Ray; Gopal C.

h e b b g e e e f c e c ghe e ge

ATTY-AGENT-FIRM: Fenwick & West LLP

ABSTRACT:

A mechanism is provided for reordering bus transactions to increase bus utilization in a computer system in which a split-transaction bus is bridged to a single-envelope bus. In one embodiment, both masters and slaves are ordered, simplifying implementation. In another embodiment, the system is more loosely coupled with only masters being ordered. Greater bus utilization is thereby achieved. To avoid deadlock, transactions begun on the split-transaction bus are monitored. When a combination of transactions would, if a predetermined further transaction were to begin, result in deadlock, this condition is detected. In the more tightly coupled system, the predetermined further transaction, if it is requested, is refused, thereby avoiding deadlock. In the more loosely-coupled system, the flexibility afforded by unordered slaves is taken advantage of to, in the typical case, reorder the transactions and avoid deadlock without killing any transaction. Where a data dependency exists that would prevent such reordering, the further transactions is killed as in the more tightly-coupled embodiment. Data dependencies are detected in accordance with address-coincidence signals generated by slave devices on a cache-line basis. In accordance with a further optimization, at least one slave device (e.g., DRAM) generates page-coincidence bits. When two transactions to the slave device are to the same address page, the transactions are reordered if necessary to ensure that they are executed one after another without any intervening transaction. Latency of the slave is thereby reduced.

19 Claims, 27 Drawing figures

h e b b g e e e f c e c ghe e ge

L7: Entry 7 of 11 File: USPT Nov 30, 1999

DOCUMENT-IDENTIFIER: US 5996036 A
TITLE: Bus transaction reordering in a computer system having unordered slaves

Abstract Text (1):
A mechanism is provided for reordering bus transactions to increase bus utilization in a computer system in which a split-transaction bus is bridged to a single-envelope bus. In one embodiment, both masters and slaves are ordered, simplifying implementation. In another embodiment, the system is more loosely coupled with only masters being ordered. Greater bus utilization is thereby achieved. To avoid deadlock, transactions begun on the split-transaction bus are monitored. When a combination of transactions would, if a predetermined further transaction were to begin, result in deadlock, this condition is detected. In the more tightly coupled system, the predetermined further transaction, if it is requested, is refused, thereby avoiding deadlock. In the more loosely-coupled system, the flexibility afforded by unordered slaves is taken advantage of to, in the typical case, reorder the transactions and avoid deadlock without killing any transaction. Where a data dependency exists that would prevent such reordering, the further transactions is killed as in the more tightly-coupled embodiment. Data dependencies are detected in accordance with address-coincidence signals generated by slave devices on a cache-line basis. In accordance with a further optimization, at least one slave device (e.g., DRAM) generates page-coincidence bits. When two transactions to the slave device are to the same address page, the transactions are reordered if necessary to ensure that they are executed one after another without any intervening transaction. Latency of the slave is thereby reduced.

Brief Summary Text (20):
A mechanism is provided for reordering bus transactions to increase bus utilization in a computer system in which a split-transaction bus is bridged to a single-envelope bus. In one embodiment, both masters and slaves are ordered, simplifying implementation. In another embodiment, the system is more loosely coupled with only masters being ordered. Greater bus utilization is thereby achieved. In accordance with one embodiment of the invention, a queuing structure includes multiple master queues and multiple slave queues. The queuing structure receives bus grant signals and respective slave acknowledge signals from respective slave devices. Each time an address bus grant is issued a record is entered in the queuing structure, the record comprising a first entry in a master queue identified by the address bus grant signals, and a second entry in a slave queue identified by the slave acknowledge signals. The first entry identifies a target slave device in accordance with the slave acknowledge signals, and the second entry identifies an originating master device in accordance with the address bus grant signals. A matching circuit is responsive to queue entries from the queuing structure for producing match bits identifying selected records the first entry of which is at the head of a master queue. A data arbitration circuit is responsive to the match bits and to queue entries from the queuing structure for generating data bus grant signals for the master devices and for generating for each slave device a multibit signal which when active identifies a transaction within the transaction queue of the slave device.

Detailed Description Text (19):
The manner in which the foregoing signals are used to decouple address tenures and

h e b b cg b cc e

data tenure may be appreciated with reference to FIG. 5. For simplicity, the address arbitration phase has not been illustrated. The address transfer phase is essentially the same as in the conventional case. The address termination phase, however, differs. The addressed slave asserts the AACK signal in the conventional manner, the AACK signal being used by the master. In parallel with AACK, the addressed slave generates a SACK signal for use by the arbiter 600. The arbiter uses this information about which slave has acknowledged in order to reorder transactions on the system bus 204.

Detailed Description Text (147):
In the case of some slave devices, the average latency of the slave device may be reduced by reordering transactions involving the slave device. In the case of DRAM, for example, page mode reads take less time than non-paged reads. Hence, in the embodiment of FIG. 19, the ArbDatSM 604' further receives page coincidence (PC) signals from at least one slave device, i.e., DRAM. The ArbDatSM block 604' receives from the slave device a separate signal for every possible pair of queue entries within the slave device, indicating whether the targets of both transactions queued within the pair of queue entries are within the same page. The ArbDatSM block 604' therefore receives Q(Q+1)/2 total page coincidence bits or, in the illustrated embodiment, 2.times.3/2=3 bits.

Detailed Description Text (165):
As previously described in relation to FIG. 6, slave ordering is a major cause of deadlock. When what would otherwise be a deadlocking transaction is detected, it is "killed" by issuing an ARtry signal. Without slave ordering, a large proportion of what would otherwise be deadlocking transactions, instead of being killed, can now be accepted and reordered in relation to other transactions so as to avoid deadlock. Such reordering is not possible, however, when a data dependency exists. For example, a read of one data location by one device followed by a write of the same data location by another device does not yield the same result as if the execution order is reversed. If a deadlock situation cannot be avoided by transaction reordering because of a data dependency, the need remains to kill the deadlocking transaction.

Detailed Description Text (177):
Referring to FIG. 25, the inputs and outputs of the ARtryGen block 613' are illustrated in greater detail. The inputs along the top and left edges of the ARtryGen block 613' remain unchanged compared to the ARtryGen block 613 of FIG. 6. Unlike the ARtryGen block 613 of FIG. 6, however, the ARtryGen block 613', instead of generating ARtry based on the assumption of ordered slaves, uses certain deadlock address-coincidence (DLAC) inputs received at the bottom edge of the block to generate a "qualified" ARtry signal only when a data dependency prevents transactions from being reordered so as to avoid the deadlock. The slave devices each monitor each system bus address tenure and compare the address placed on the bus to addresses queued within the respective slave devices. If the address on the bus is the same as an address already queued within the slave device, the slave device raises its DLAC signal to the ARtryGen block 613'. All slave devices or only selected slave devices (most importantly DRAM) may monitor the bus and signal the ARtryGen block 613' in this manner. In the illustrated embodiment, all slave devices are assumed to provide a DLAC signal. The ARtryGen block 613' therefore receives signals DLAC.sub.0 through DLAC.sub.(S+1).

Current US Original Classification (1):
710/110

Current US Cross Reference Classification (1):
709/208

Current US Cross Reference Classification (2):
710/107

h e b b cg b cc e

CLAIMS:

1. In a computer system having a system bus and having arbitration circuitry, multiple master devices including a system microprocessor, and multiple slave devices, all coupled to the system bus, a method of reordering system bus transactions, comprising the steps of:

receiving and queuing within a particular slave device a plurality of transactions;

within said arbitration circuitry, arbitrating between pending transactions based on arbitration policies including an arbitration policy that responses are received by respective master devices in the same order as requests were issued by the respective master devices; and

at least some of the time, said arbitration circuitry, without signalling said microprocessor, signalling said particular slave device such that the system bus is granted for a later queued transaction within said particular slave device prior to being granted for an earlier queued transaction.

h e b b cg b cc e

US00RE38428E

(19) **United States**
(12) **Reissued Patent**
Kelly et al.

(10) **Patent Number: US RE38,428 E**
(45) **Date of Reissued Patent: Feb. 10, 2004**

(54) **BUS TRANSACTION REORDERING IN A COMPUTER SYSTEM HAVING UNORDERED SLAVES**

(75) Inventors: **James D. Kelly**, Scotts Valley, CA (US); **Michael L. Regal**, Pleasanton, CA (US)

(73) Assignee: **Apple Computer, Inc.**, Cupertino, CA (US)

(*) Notice: This patent is subject to a terminal disclaimer.

(21) Appl. No.: 10/006,939

(22) Filed: Nov. 30, 2001

Related U.S. Patent Documents

Reissue of:
(64) Patent No.: 5,996,036
Issued: Nov. 30, 1999
Appl. No.: 08/779,632
Filed: Jan. 7, 1997

U.S. Applications:
(63) Continuation-in-part of application No. 08/432,622, filed on May 21, 1995, now abandoned.

(51) Int. Cl.[7] G06F 9/46; G06F 13/36; G11C 7/00

(52) U.S. Cl. 710/110; 710/107; 709/208; 370/402

(58) Field of Search 710/110, 107, 710/263, 41, 52, 311; 709/100, 208; 714/47; 711/151; 370/402

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,181,974	A	1/1980	Lemay et al.
4,473,880	A	9/1984	Budde et al.
4,494,193	A	1/1985	Brahm et al.
4,965,716	A	10/1990	Sweeney
5,006,982	A	4/1991	Ebersole et al.
5,191,649	A	3/1993	Cadambi et al.
5,257,356	A	10/1993	Brockmann et al.
5,287,477	A	2/1994	Johnson et al.
5,305,442	A	4/1994	Pedersen et al.
5,307,505	A	4/1994	Houlberg et al.
5,327,538	A	7/1994	Hamaguchi et al.
5,327,570	A	7/1994	Foster et al.
5,333,276	A	7/1994	Solari
5,345,562	A	9/1994	Chen

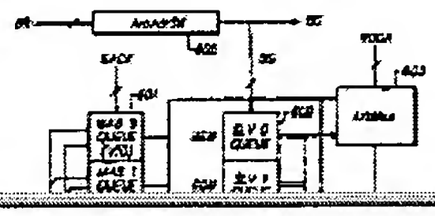
(List continued on next page.)

Primary Examiner—Gopal C. Ray
(74) *Attorney, Agent, or Firm*—Fenwick & West LLP

(57) **ABSTRACT**

A mechanism is provided for reordering bus transactions to increase bus utilization in a computer system in which a split-transaction bus is bridged to a single-envelope bus. In one embodiment, both masters and slaves are ordered, simplifying implementation. In another embodiment, the system is more loosely coupled with only masters being ordered. Greater bus utilization is thereby achieved. To avoid deadlock, transactions begun on the split-transaction bus are monitored. When a combination of transactions would, if a predetermined further transaction were to begin, result in deadlock, this condition is detected. In the more tightly coupled system, the predetermined further transaction, if it is requested, is refused, thereby avoiding deadlock. In the more loosely-coupled system, the flexibility afforded by unordered slaves is taken advantage of to, in the typical case, reorder the transactions and avoid deadlock without killing any transaction. Where a data dependency exists that would prevent such reordering, the further transactions is killed as in the more tightly-coupled embodiment. Data dependencies are detected in accordance with address-coincidence signals generated by slave devices on a cache-line basis. In accordance with a further optimization, at least one slave device (e.g., DRAM) generates page-coincidence bits. When two transactions to the slave device are to the same address page, the transactions are reordered if necessary to ensure that they are executed one after another without any intervening transaction. Latency of the slave is thereby reduced.

19 Claims, 21 Drawing Sheets



US-PAT-NO: 6505259

DOCUMENT-IDENTIFIER: US 6505259 B1

TITLE: Reordering of burst data transfers across a host bridge

---------- KWIC ---------

Detailed Description Text - DETX (11):

FIG. 3 also shows queue 240 to receive an entire line of read data from agent or device 180, 185, or 190. In one example, queue 240 is a line-size buffer. As noted above, address segmentation unit 220 stores the address bits for ordering the line of read data according to the transaction requested by processor 100. Once the entire line of read data is present in queue 240, steering logic, for example, in state machine 230 is employed to reconfigure data based on the address transaction requested. If the transaction requested was a linear line read (e.g., A[4:3]# is 00b), the read data is returned as linear line data. Conversely, if the transaction requested is a non-linear line read (e.g., A[4:3]# is 01b, 10b, or 11b), the configuring address is associated with the read data and the data returned as non-linear line data. One way this modification may be done is by utilizing multiplexer 250 coupled to queue 240 and controlling the output to processor 100. For example, a state machine may be utilized such that when a transaction is linear, multiplexor 250 does not reorder the line data. When the transaction is non-linear, state machine 230 utilizes mutliplexer 250 to reorder the line data prior to forwarding to the processor.

US-PAT-NO: 6246993

DOCUMENT-IDENTIFIER: US 6246993 B1

TITLE: Reorder system for use with an electronic printing press

---------- KWIC ---------

Detailed Description Text - DETX (192):

The local processor 514 continues looping through the flowcharts illustrated in FIGS. 14c and 14d, immediately reordering errored books where appropriate (blocks 678 and 680); sending new reorder messages to the global reorder processor 510 when necessary (blocks 678 and 682); requesting global reorders after every good book that represents the end of a carrier route or change in the postal code when no global reorders are pending in the local queue (i.e., the global reorder flag is false) (blocks 636, 656, 658, and 660); and obtaining new GRT_Seg_IDs from the global reorder processor whenever a sub-job is completed (block 674) until the press controller 514 determines that the last book in the book ticket and the last global reorder book sent from the global reorder processor 510 have been printed (block 630). At such a point, control advances to block 631, and the local processor 514 waits for an operator command to pull a new book ticket to restart the process, or to call the process outlined in FIG. 19 and discussed above.

US005293620A

United States Patent [19]

Barabash et al.

[11] Patent Number: 5,293,620

[45] Date of Patent: Mar. 8, 1994

[54] METHOD APPARATUS FOR SCHEDULING TASKS IN REPEATED ITERATIONS IN A DIGITAL DATA PROCESSING SYSTEM HAVING MULTIPLE PROCESSORS

[75] Inventors: William Barabash, Acton; William S. Yerazunis, Hudson, both of Mass.

[73] Assignee: Digital Equipment Corporation, Maynard, Mass.

[21] Appl. No.: 927,785

[22] Filed: Aug. 10, 1992

Related U.S. Application Data

[63] Continuation of Ser. No. 366,774, Jun. 15, 1989, abandoned.

[51] Int. Cl.[5] G06F 9/40

[52] U.S. Cl. 395/650; 364/DIG. 1; 364/230.3; 364/254.5; 364/262.1; 364/264.6; 364/281.3; 364/281.8

[58] Field of Search 395/650

[56] References Cited

U.S. PATENT DOCUMENTS

4,229,790	10/1980	Gilliland et al.	364/200
4,286,322	8/1981	Hoffman et al.	364/200
4,394,727	6/1983	Hoffman et al.	364/200
4,893,638	5/1989	Vallaro	364/900

OTHER PUBLICATIONS

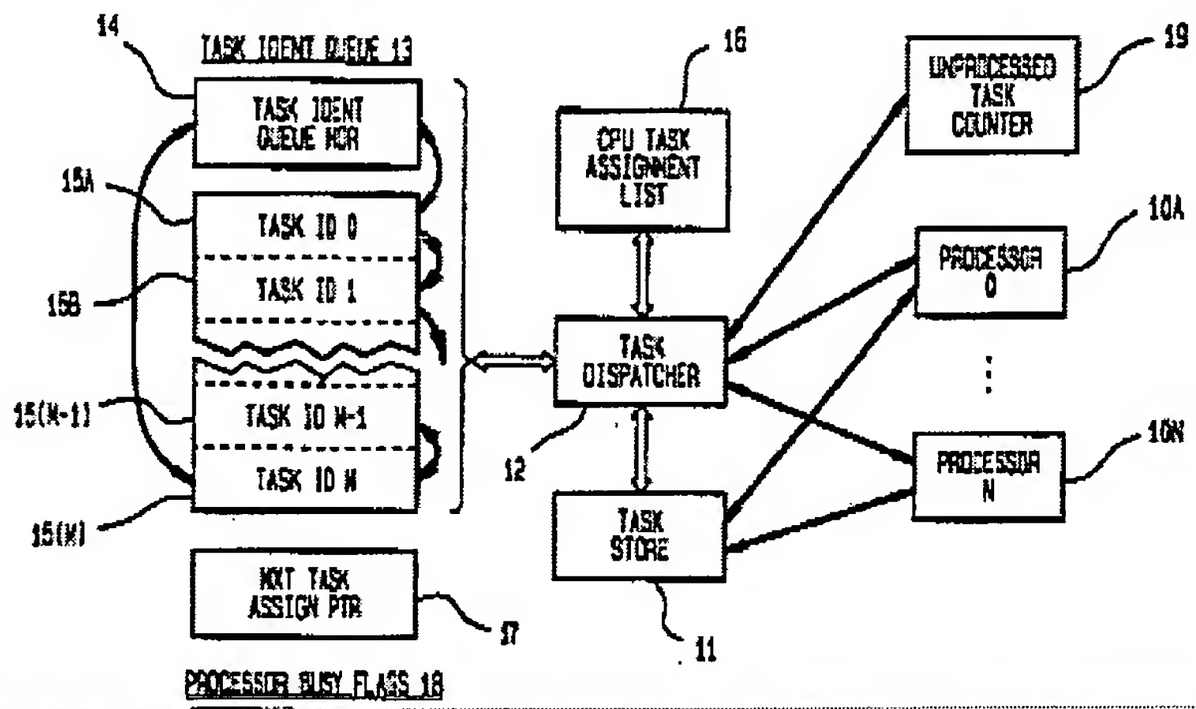
D. Knuth, The Art of Computer Programming, vol. 3, Sorting and Searching, pp. 397-399.

Primary Examiner—Gareth D. Shaw
Assistant Examiner—Kakali Chaki
Attorney, Agent, or Firm—Kenyon & Kenyon

[57] ABSTRACT

A task dispatch system for use in connection with a plurality of digital data processors for processing tasks. The task dispatch system maintains a task identification queue including a plurality of task identification entries defining a series of tasks to be processed during an iteration. A task dispatcher dispatches tasks to the processors in the order defined by the task identification queue. At the end of an iteration, the task dispatcher reorganizes the entries in the task identification queue so as to dispatch one or more tasks, which were completed last during an iteration, at the beginning of a subsequent iteration.

51 Claims, 8 Drawing Sheets



L1: Entry 1 of 1 File: USPT Feb 10, 2004

DOCUMENT-IDENTIFIER: US RE38428 E
TITLE: Bus transaction reordering in a computer system having unordered slaves

Brief Summary Text (12):
In a traditional pipelined implementation, data bus tenures are kept in strict order with respect to address tenures. However, external hardware can further decouple the address and data buses, allowing the data tenures to occur out of order with respect to the address tenures. Second-generation PowerPC computers include computers whose architecture was especially designed for high performance and that incorporated such hardware. This architecture supports true split-bus operation with ordered slaves and ordered masters. "Ordered" means each master and each slave has its own independent FIFO structure supporting "ordered" service to transactions posted to it. If a slave receives three transactions A, B, and C, then it will respond to A first, B second, and C third. If a master performs transactions D, E, and F, then it expects servicing of those transactions in the order of D first, E second, and F third. There can be up to a selected number of outstanding master/slave pair transactions in the architecture at one time. In one preferred embodiment, this selected number is three outstanding pair transactions. As a result, in the foregoing architecture, an expansion bridge may concurrently have one outstanding slave transaction to it and one outstanding master transaction from it. Although ordered masters and slaves, as opposed to unordered masters and slaves, provide an overall simplification to system architecture, they can lead to deadlocks when there are conflicting completion dependencies.

Detailed Description Text (15):
The architecture of the computer system of FIG. 2 decouples address and data tenures such that data bus utilization is increased. This increase in data bus utilization allows for higher real-time performance to be achieved. In particular, the present invention allows for a true split-bus architecture with ordered slaves and ordered masters. "Ordered," in one usage, means each master and each slave has its own independent FIFO structure supporting "ordered" service to transactions posted to it. If a slave receives three transactions A, B, and C, the it will respond to A first, B second, and C third. If a master performs transactions D, E, and F, then it expects servicing of those transactions in the order of D first, E second, and F third. In one embodiment, there can be up to three outstanding master/slave pair transactions at one time.

Detailed Description Text (19):
The manner in which the foregoing signals are used to decouple address tenures and data tenure may be appreciated with reference to FIG. 5. For simplicity, the address arbitration phase has not been illustrated. The address transfer phase is essentially the same as in the conventional case. The address termination phase, however, differs. The addressed slave asserts the AACK signal in the conventional manner, the AACK signal being used by the master. In parallel with AACK, the addressed slave generates a SACK signal for use by the arbiter 600. The arbiter uses this information about which slave has acknowledged in order to reorder transactions on the system bus 204.

h e b b g e e e f c e e ge

Detailed Description Text (43):
Assume now that an RDDA is received from Slave 3. Transactions 1, 2 and 3 will then, in that order, be granted the bus and will complete. In the foregoing example, whereas the address order of the transactions is 1, 2, 3, 4, 5, the data order is 4, 5, 1, 2, 3.

Detailed Description Text (59):
The PCI2PCI bridge has become interlocked, and must flush a posted write upstream of itself; in this case the write is headed toward the ARBus, and Expansion Bridge 1's buffers are full and cannot currently accept the write. The first two outstanding transactions in this scenario are 1) Master Processor 1 has an outstanding read of Expansion Bridge 1, followed by 2) Master Processor 1 has an outstanding write to Expansion Bridge 1. The third attempted transaction is a write cycle from Expansion Bridge 1 to main memory. However, this write cycle is to copyback-cacheable space and causes a snoop hit in Processor 1's cache. Processor 1 retries Expansion Bridge 1's write cycle, but now needs to push the dirty cache line to main memory. However, at this point it is unable to push the dirty cache line due to its outstanding write to Expansion Bridge 1. With the use of DBWO*, Processor 1 could have re-ordered the snoop push write transaction around its outstanding read of Expansion Bridge 1 (transaction number 1). However, the MPC60x microprocessor is not capable of re-ordering the snoop push write transaction around its own outstanding write. This is a Type-B LockUp, caused by Processor 1's inability to complete its read due to the PCI2PCI bridge's interlocking behavior. This deadlock is avoided by having the ARBus arbiter prevent the Processor from writing to an expansion bridge if it has an outstanding read of the expansion bridge. This will allow the Processor to perform the Snoop Push write transaction if required.

Detailed Description Text (64):
Following ARBus protocol after an ARTRY*, all masters on the bus deassert their Bus Requests to give the Processor a guaranteed window being the only bus requestor. This guarantees that the Processor, who normally has lowest ARBus priority, acquires the bus next in order to complete a high priority transaction such as a Snoop Push. In this case, the ARBus protocol causes the lower priority expansion bridge to never receive a Bus Grant due to the higher priority Video requesting the ARBus to complete its access. Since the completion of the Video access is dependent on the expansion bridge freeing up some buffer space, and since the expansion bridge must get the ARBus to complete its access or receive an ARTRY* in order to free up PCI Bus 1 to free up buffer space for the Video write to come in, the expansion bridge effectively needs higher priority than Video this time. This is a Type-C LockUp, and is avoided by having an expansion bridge keep its Bus Request asserted the clock following an ARTRY* if it is the source of the ARTRY*. This is precisely the protocol the MP60X processor performs to effectively achieve a higher priority when necessary.

Detailed Description Text (97):
The description thus far has assumed a system in which both masters and slaves are ordered. In particular, the 60X microprocessor assumes that its transactions are ordered. As a consequence, master ordering is to some extent ingrained within the underlying system architecture. Slave ordering, on the other hand, although it may be convenient from an implementation perspective, is not required. Increased efficiency may be achieved by relaxing the constraint of slave ordering, thereby allowing transaction independence within slaves. To achieve unordered slaves, additional information must be exchanged between the slaves and the arbiter. As before, this information may be exchanged in the form of additional side-band signals not provided for by the MPC60X bus specification.

Detailed Description Text (99):
Considering first the block ArbMux 603', in order to allow for transaction independence within slaves, the ArbMux 603' receives as inputs all of the queue

h e b b g e e e f c e e ge

entries of all of the slave queues (instead of just all of the front entries as in FIG. 6). Therefore, if the masters are numbered 0 through M, the slaves are numbered 0 through S and the queues locations within each slave queue are numbered 0 through Q, then the ArbMux 603' receives (S+1)(M+1+1)(Q+1) bits of information from the slave queues. One of the bits in the expression (M+1+1) is a valid bit that allows for a flop-based queue implementation instead of one requiring random-access memory. In an exemplary embodiment with S=5, M=4, and Q=2, the number of bits received from the slave queues is 6.times.6.times.3=108 bits. Since masters remained ordered, the ArbMux 603' continues to receive only the front entries from the master queues, the same as in FIG. 6. In the illustrated embodiment, the ArbMux 603' receives from the master queues (M+1)(S+1+1+1)=5.times.8=40 bits. As in the case of the slave queue entries, one of the bits in the expression (S+1+1+1) is a valid bit. The extra bit in the expression (S+1+1+1) is a read/write bit as described previously.

Detailed Description Text (107):
In FIG. 6, a transaction is allowed to proceed only if it is the frontmost transaction of both the master and the slave. The matching queue location within the slave is by definition always the frontmost valid queue location within the slave. In the case of ArbMux 603 of FIG. 6, therefore, its function is to identify masters whose next transaction in order is also the next transaction in order of the target slave device. In the case of the ArbMux 603' of FIG. 19, slave ordering is no longer required. Hence, the function of the ArbMux 603' is to identify for each master the queue location within the target slave that matches the frontmost transaction of the master. The ArbMux 603' also indicates whether transaction data for that queue location is ready. Hence, for each master, two bits, a SlvMatch bit and a SlvRdReady bit, are output for each queue location. In the case of master M.sub.0, the bit pairs output by the ArbMux 603' are designated M.sub.0 Q.sub.0, M.sub.0 Q.sub.1, . . . , M.sub.0 Q.sub.(Q+1), and likewise for each succeeding master up to and including the last master M.sub.(M+1), the outputs for which are M.sub.(M+1) Q.sub.0, M.sub.(M+1) Q.sub.1, . . . , M.sub.(M+1) Q.sub.(Q+1). If a master has a valid transaction in its queue, then for the frontmost valid transaction, the SlvMatch signal for that master that corresponds to the matching target slave queue location will be asserted. If the master has no valid transaction in its queue, then no signal is asserted for that master.

Detailed Description Text (114):
As previously described in relation to FIG. 6, slave ordering is a major cause of deadlock. When what would otherwise be a deadlocking transaction is detected, it is "killed" by issuing an ARtry signal. Without slave ordering, a large proportion of what would otherwise be deadlocking transactions, instead of being killed, can now be accepted and reordered in relation to other transactions so as to avoid deadlock. Such reordering is not possible, however, when a data dependency exists. For example, a read of one data location by one device followed by a write of the same data location by another device does not yield the same result as if the execution order is reversed. If a deadlock situation cannot be avoided by transaction reordering because of a data dependency, the need remains to kill the deadlocking transaction.

Detailed Description Text (118):
To take a concrete example, assume that the frontmost queue entry for Master 0 designates Slave 0. Assume further that the SlvMatch bits for Master 0 are 010, indicating that the match is for queue entry 1 of Slave 0. Without taking into account the address coincidence bits of Slave 0, the transaction in queue entry 1 will be executed if Master 0 is the winning master. Now assume that the address coincidence bits of Slave 0 are 100, indicating that the transactions within queue locations 0 and 1 are directed to the same cache line. A data dependency therefore exists between the transactions such that they must be executed in order. To prevent the transaction in queue entry 1 from being executed before the transaction in queue entry 0, the SlvMatch bits of Master 0 are modified, i.e., changed from

h e b b g e e e f c e e ge

010 to 000. The same modification is performed for each arbitration cycle until the transaction in queue entry 0 has executed. The address coincidence bits for Slave 0 will then be 000. The SlvMatch bits of Master 0 then, instead of being modified, remain 010 such that the transaction in queue entry 1 may be executed next if Master 0 is the winning master.

Detailed Description Text (120):
In operation, the ArbDatSM 604' determines to which masters the PC bits will be applied, e.g., which masters have a DRAM transaction at the front of their queues, in accordance with the SACK vectors at the head of the master queues. The PC bits are then used to determine which queue locations cannot have the transactions queued therein go next without forfeiting the speed advantage to be gained from paged access. In practice, if a PC bit is asserted, the transactions to which the PC bit relates will be scheduled for execution prior to any other transactions involving the DRAM. In other words, if the DRAM has three transactions queued, two of which are to the same page, the execution order will be COINCIDENT, COINCIDENT, NON-COINCIDENT, instead of NON-COINCIDENT, COINCIDENT, COINCIDENT, although both sequences yield the same speed advantage. In other embodiments, any execution order that results in the page-coincident transactions being executed one after another without any intervening transaction may be acceptable for purposes of the PC bits.

h e b b g e e e f c e e ge